Secure programing assignment 1

CSE 5382-001

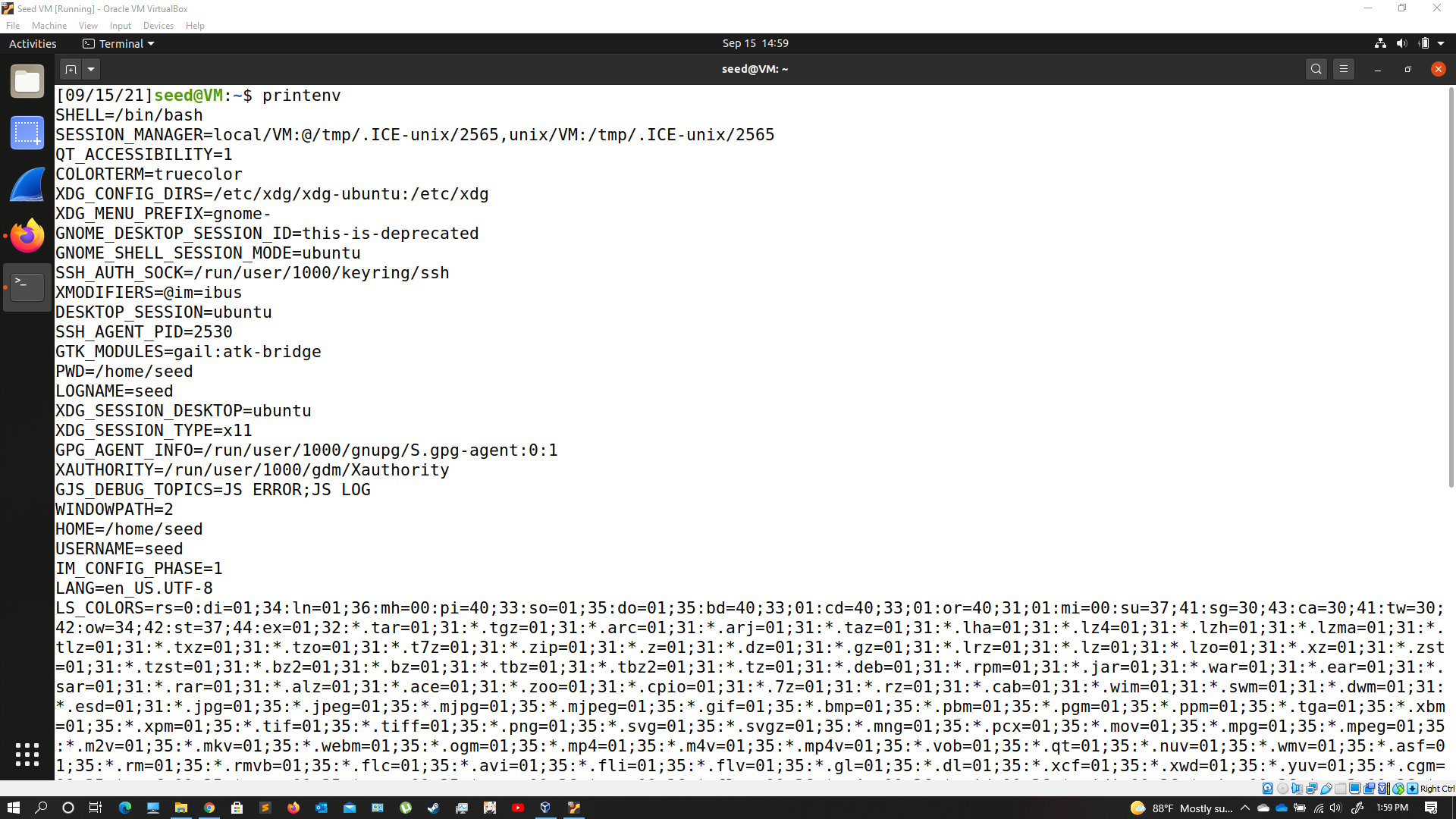
Submitted by:

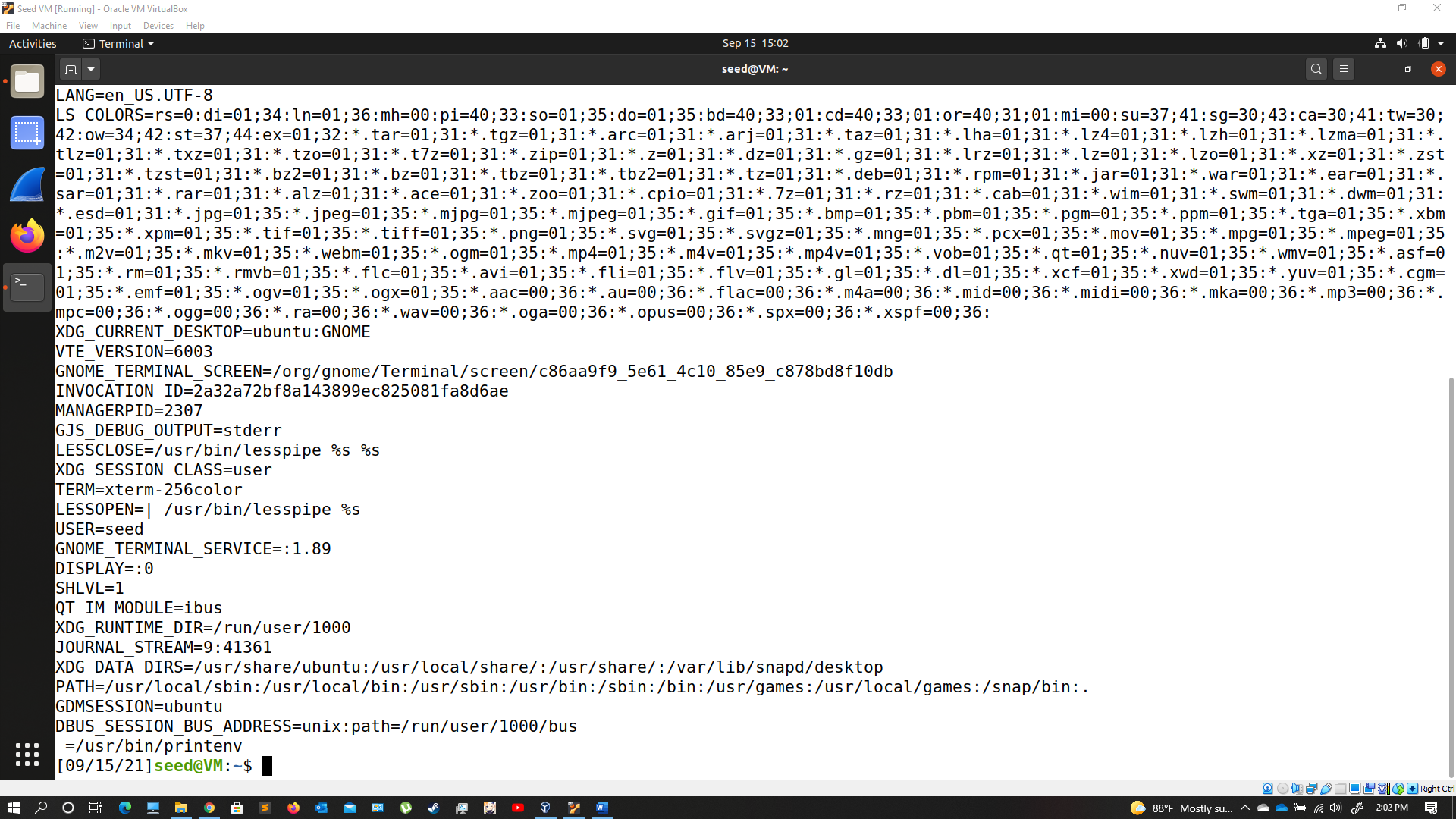
Name: Anuraag Venkatapuram Sreenivas

Student id: 1001716458

Task 1: Manipulating Environment Variables

Use “printenv” or “env” command to print out the environment variables:



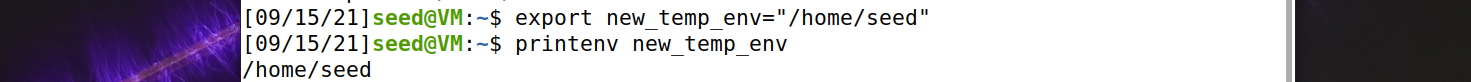


Observation: Prints all the environment variables

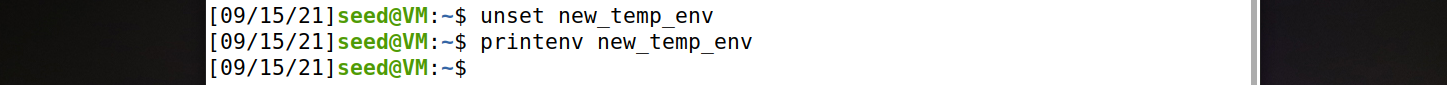


Observation: Displays PWD environment variable

Use “export” and “unset” to set or unset environment variables:



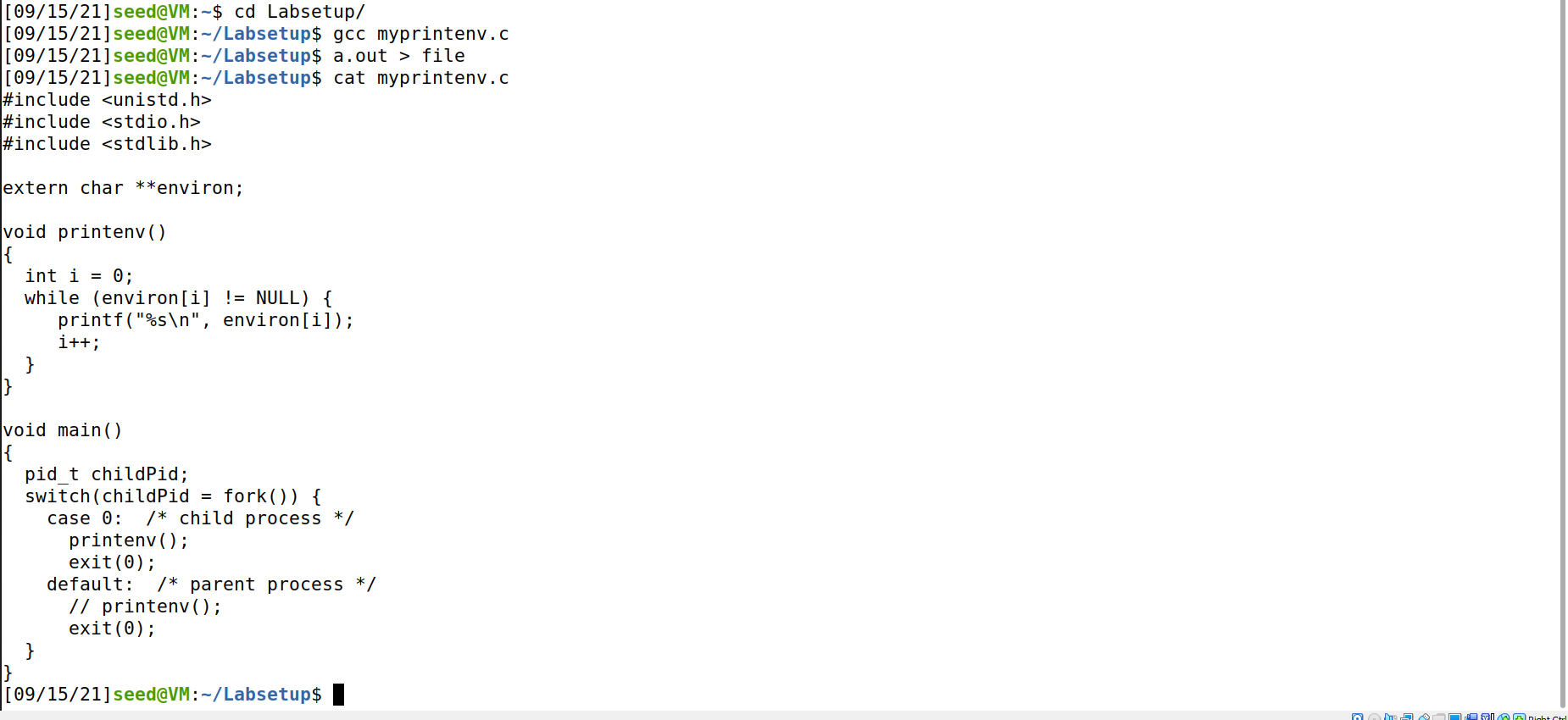
Observation: Creating an environment variable using “export” and using “printenv” command to see if the variable is created at that location.



Observation: using “unset” to remove the variable from the list.

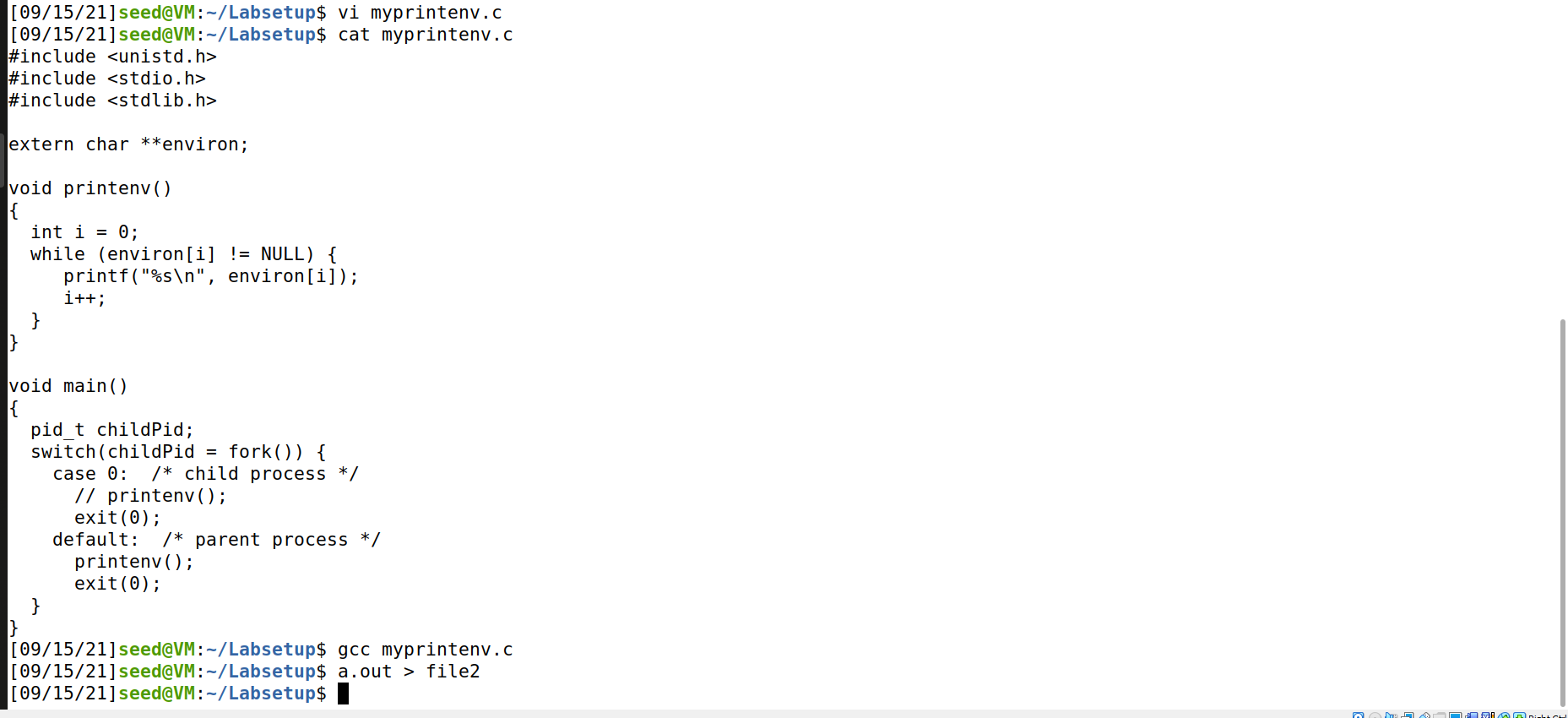
Task 2: Passing Environment Variables from Parent Process to Child Process

Step1:



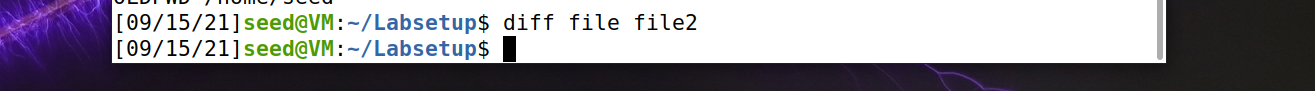
Observation: executing child process

Step2:



Observation: executing parent process

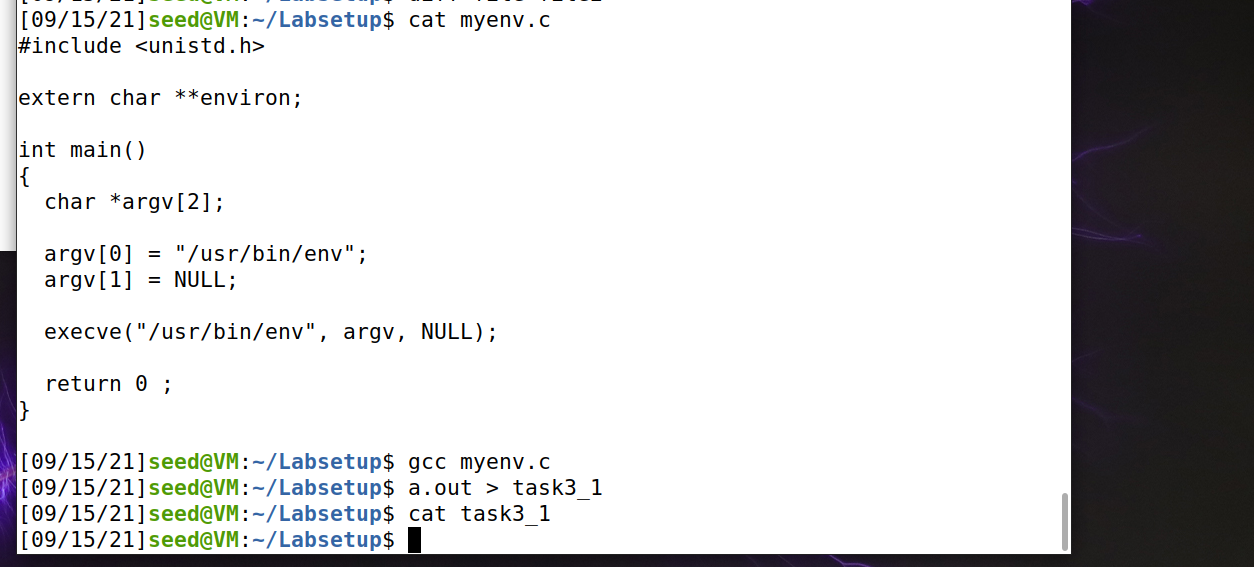
Step3:



Observation: there is no difference in the output files created in the either step’s as the “diff” command doesn’t give out any output. (can be verified using cat command on each file) Hence it can be interpreted that the child process inherits all the environment variables from parent.

Task 3: Environment Variables and execve()

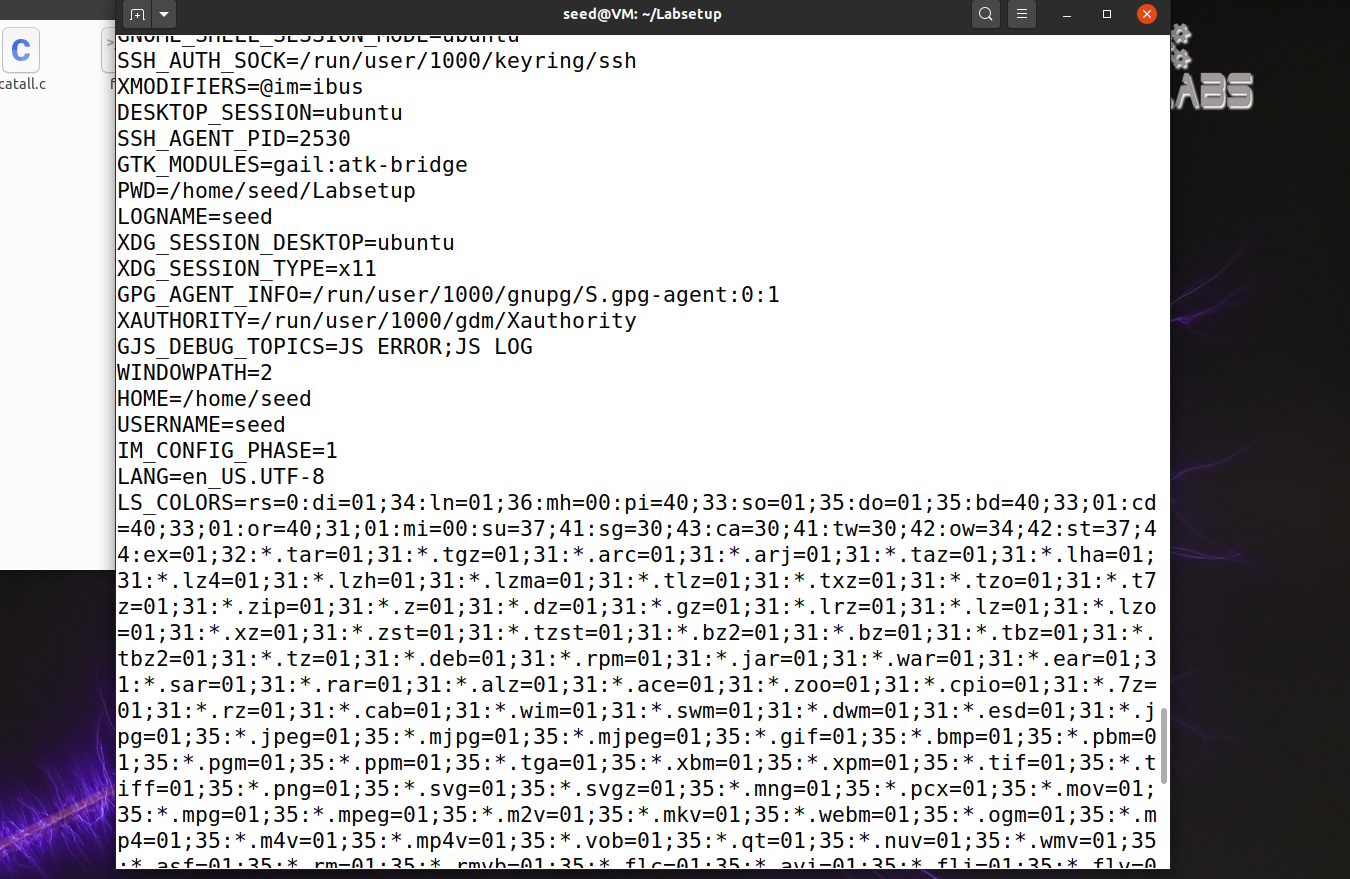
Step1:



Observation: simply executing “/usr/bin/env” but we do not get any environment variables as NULL is used as the third parameter

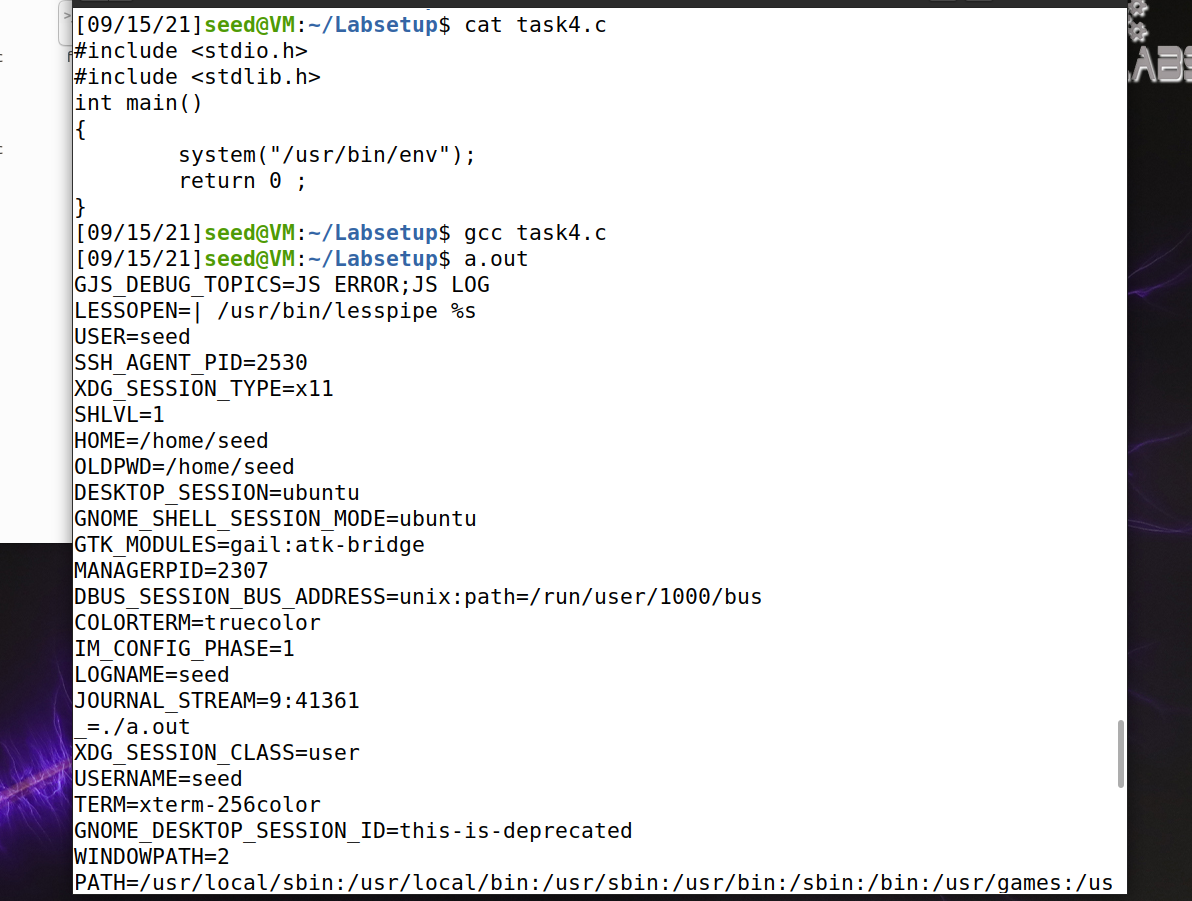
Step2:







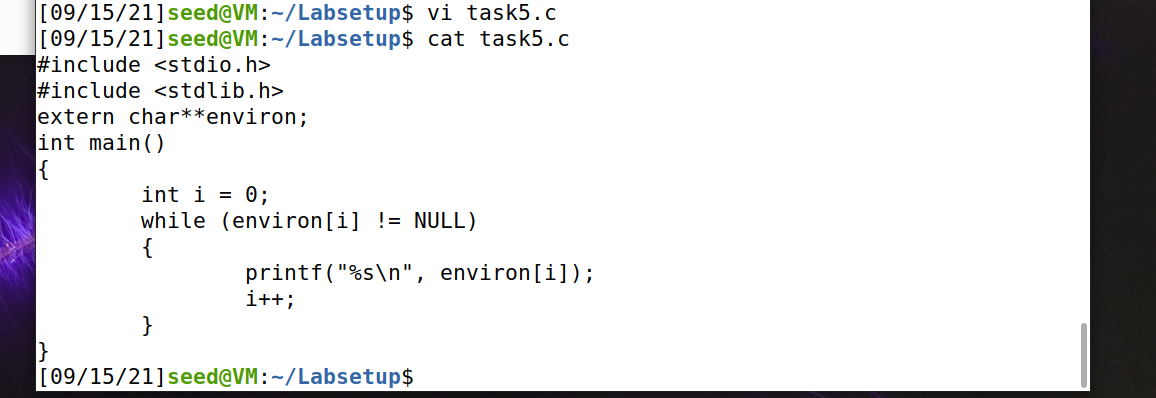
Observation: after changing the 3rd argument from null to environ and compiling and executing the same program, we can see that the process prints all the environment variables that it inherits. Here we are passing the environment variables in while invoking the execve() method along with “/usr/bin/env”.

Task 4: Environment Variables and system()

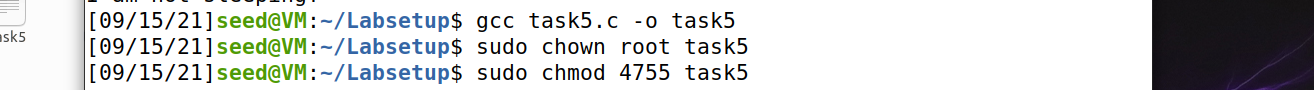
Observation: The program prints all the environment variables of the current process. The system() command would use fork() to create a child process which executes the shell command. Hence the child process would display all the environment variables.

Task 5: Environment Variable andSet-UIDPrograms:

Step1: running the given program



Step2: changing permissions



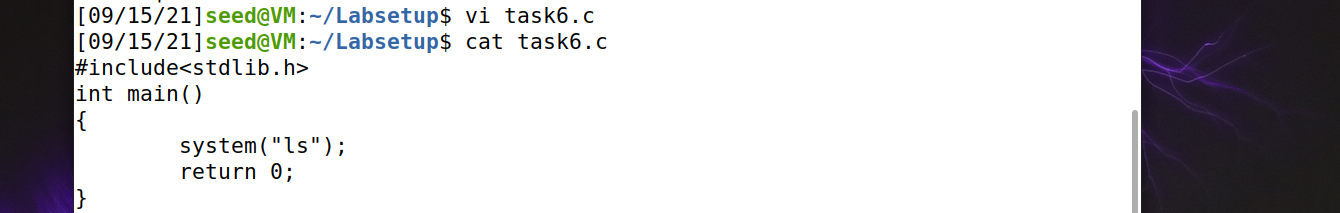
Step3:



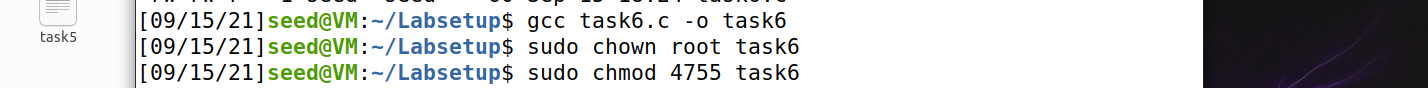


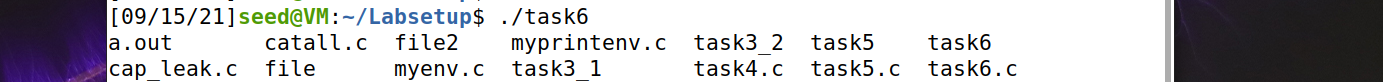
Observation: the new environment variable (shown in second image) that was created was inherited along with the other environment variables and are displayed when the program is executed.

Task 6: The PATH Environment Variable and Set-UID Programs:



Compiling the above program, and change its owner to root, and make it a Set-UID program.





Changing the code to test our own command for syterm():

Graphical user interface, text

Description automatically generated

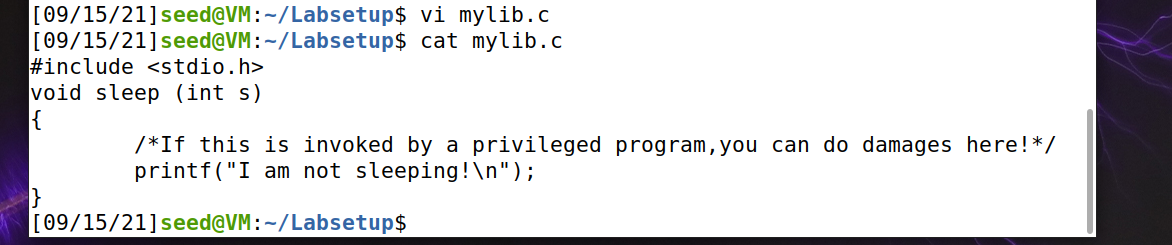
Text

Description automatically generated

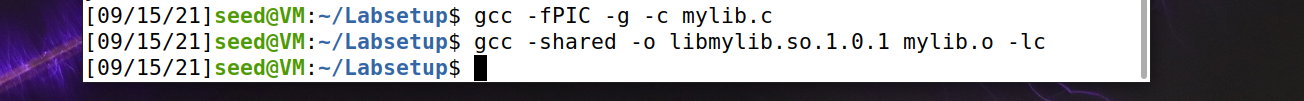
Observation: the system() command invokes fork() which creates new process and executes the command. This execution could be affected by other users since system() command would refer to the environment variables. Therefore this prgm is proven to be very dangerous.

Task 7: The LD PRELOAD Environment Variable and Set-UID Programs:

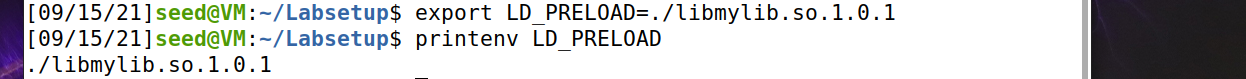
Step1.1: building a dynamic link library



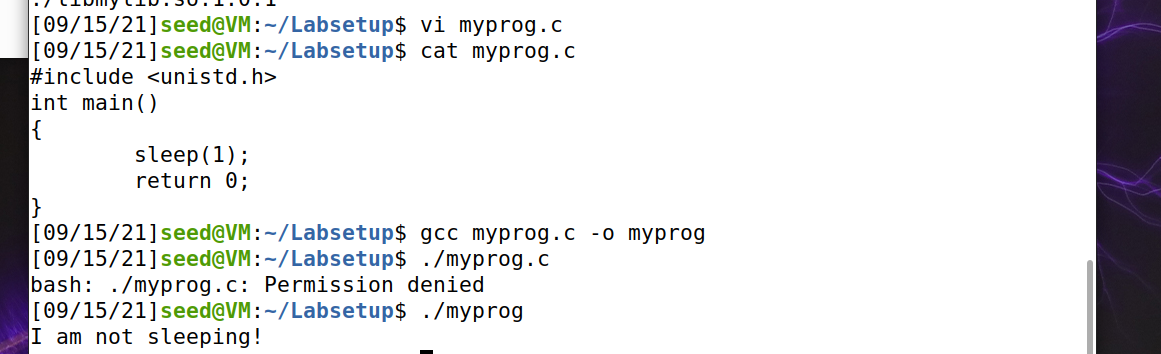
Step1.2: compiling the program



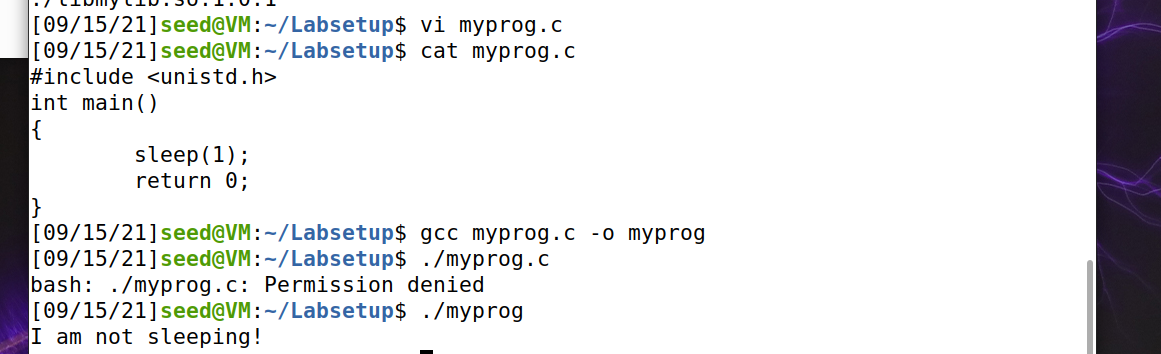
Step1.3: set the LD PRELOAD environment variable



Step1.4: compiling myprog.c

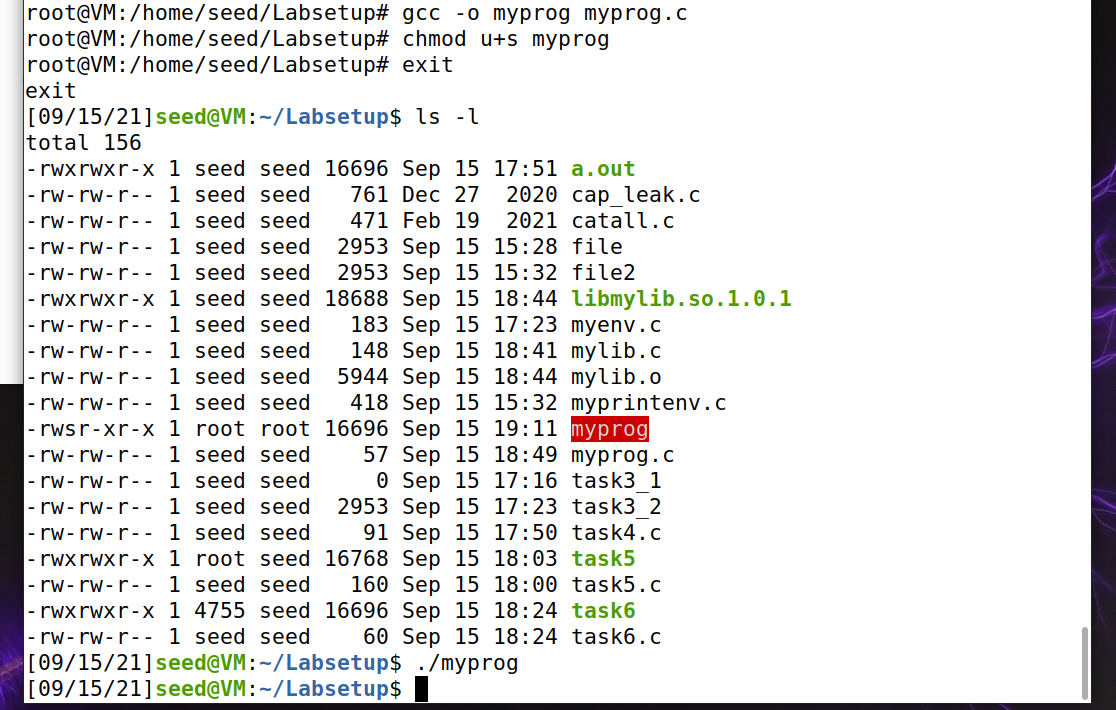


Step2.1: Make myprog regular program, and run it as a normal user.



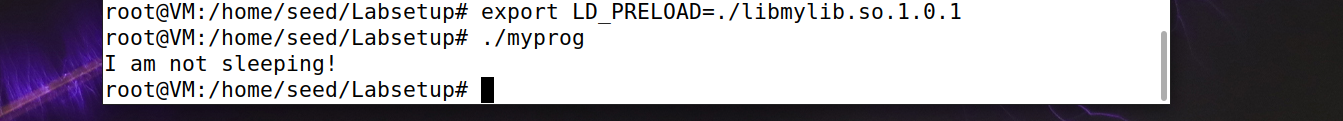
Observation: we see the custom sleep() method that we created is being invoked as we have included it in the environment variable

Step2.2: Make myprog a Set-UID root program, and run it as a normal user.



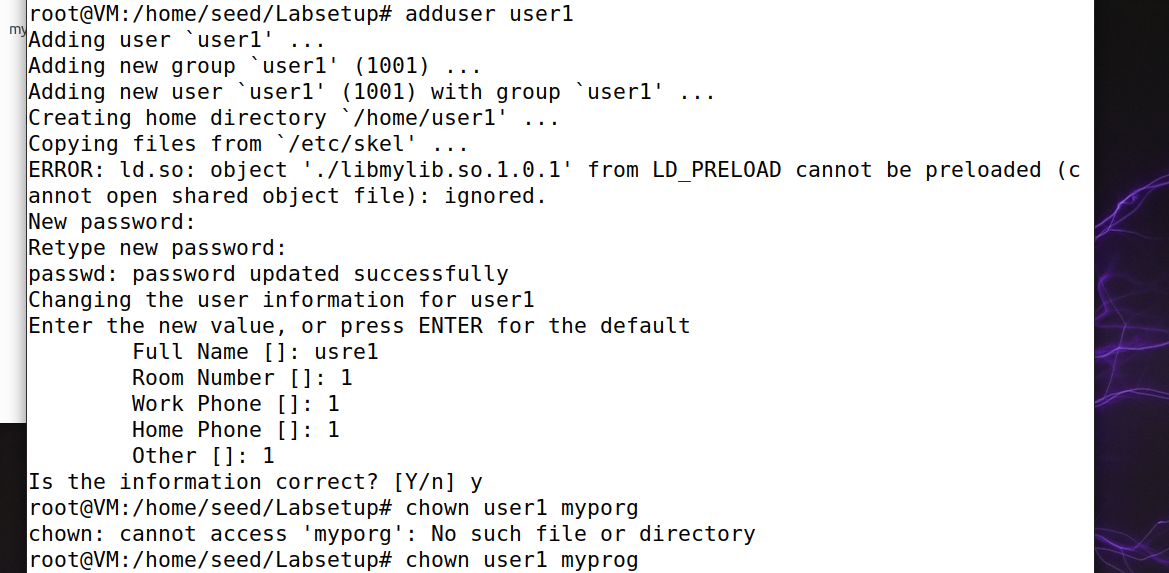
Observation: when a program is set as a privileged root program it would ignore the previously set LD\_PRELOAD variable while executing. Hence the custom sleep function is not called here.

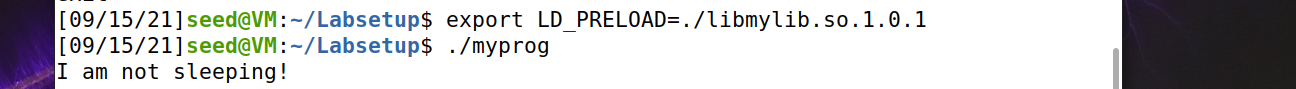
Step2.3: Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.



Observation: since the LD\_PRLOAD is set to the point to the custom library, again the privileged root program would call the custom Sleep() method.

Step2.4: Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD PRELOAD environment variable again in a different user’s account (not-root user) and run it.

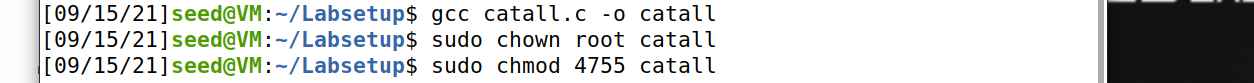


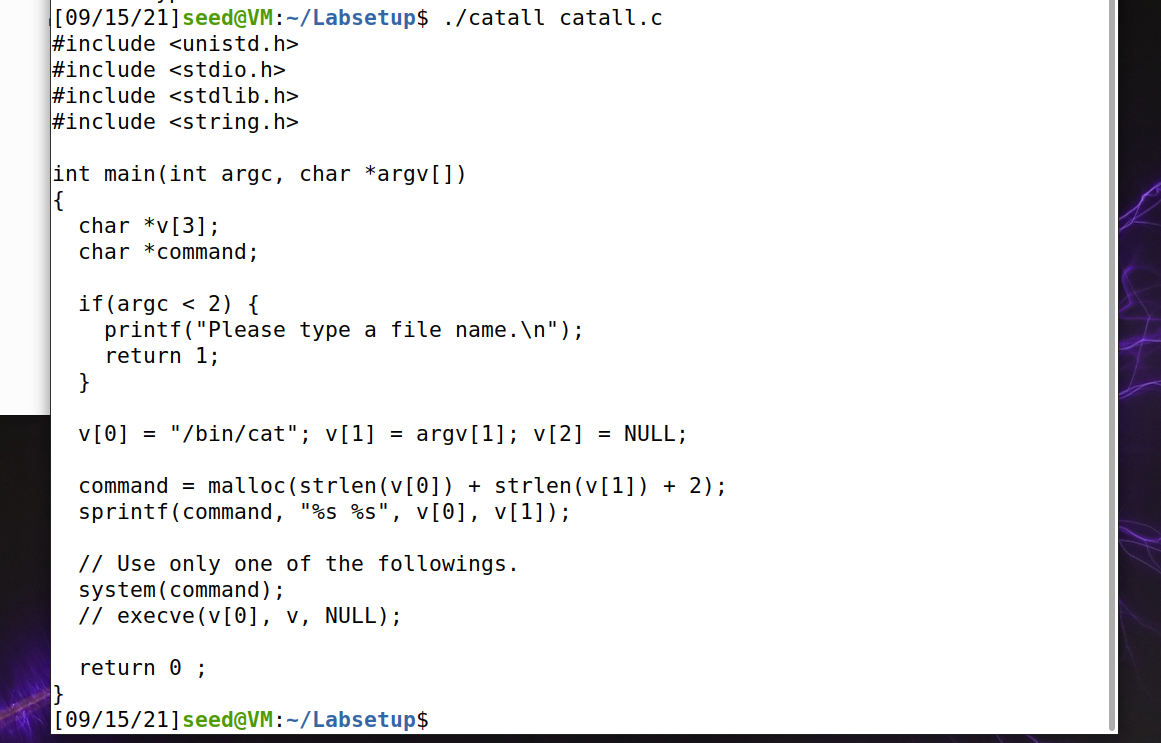


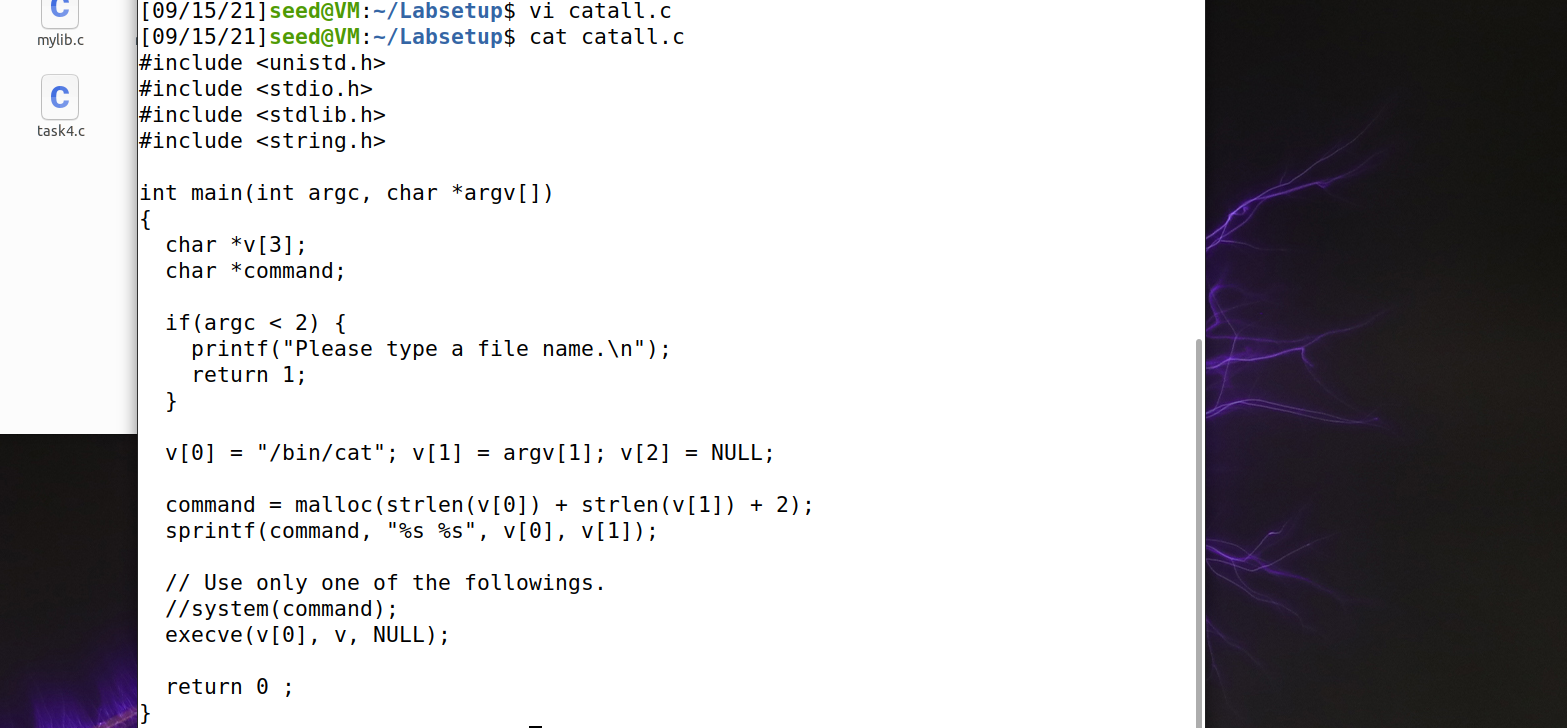
Observation: we see that out custom sleep() method is being invoked in this case.

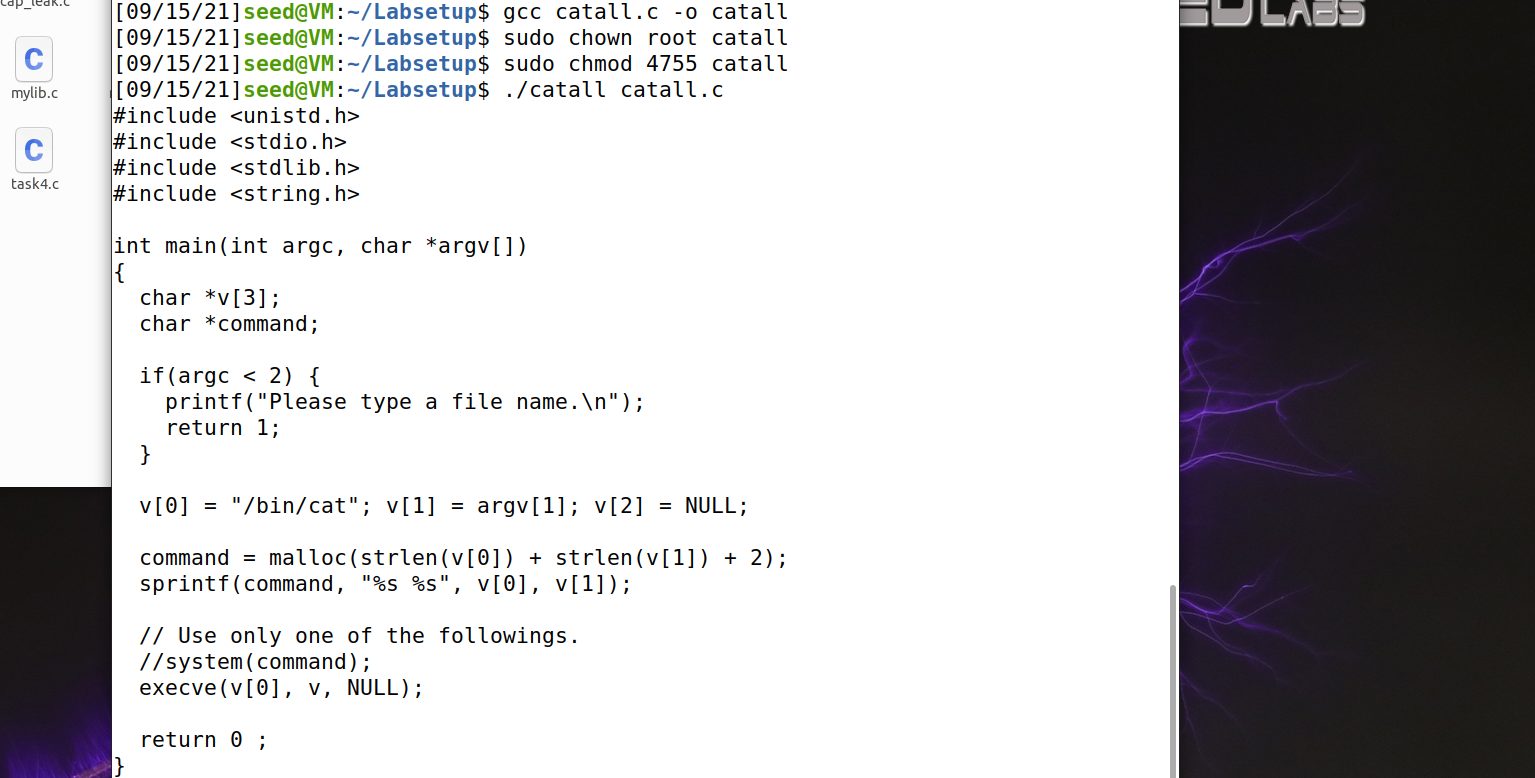
Task 8: Invoking External Programs Using system() versus execve()

Step1: Compile the above program, make it a root-owned Set-UID program. The program will use system() to invoke the command.



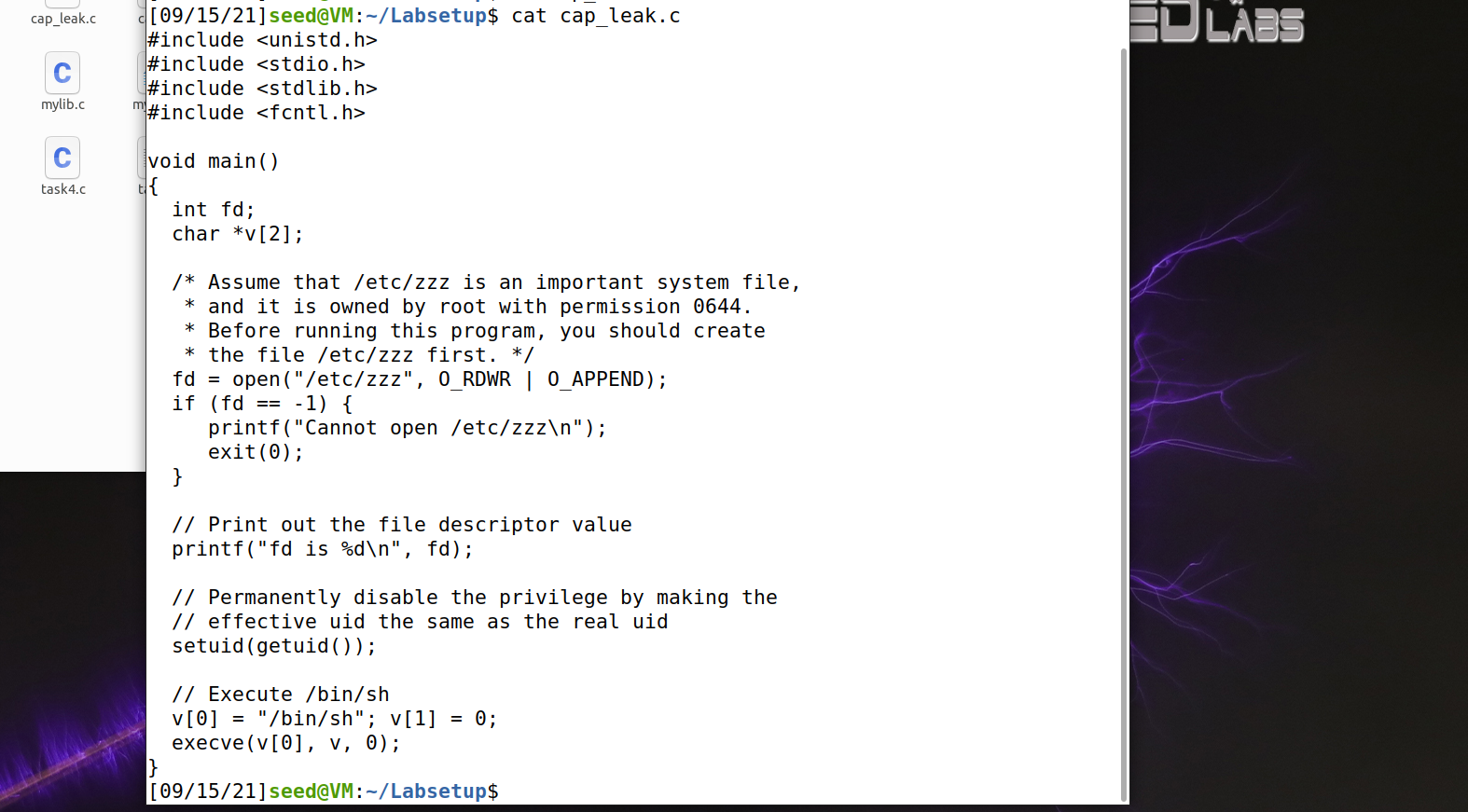


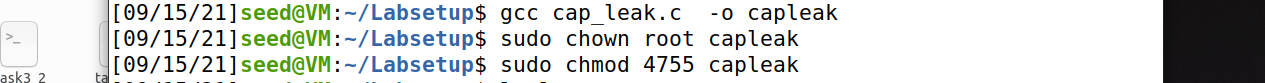
Step 2: Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command.



Observation: execve() uses root environment variables. And these variables can be set by privileged users only. Hence, using execve() does not cause any attack in step 1.

Task 9: Capability Leaking







Observation: non root users will not have write access to the file zzz. When the file owner is changed to root the current privileges are revoked making the file inaccessible.